Structured compressible flow computations on the connection machine

Stephane Lanteri, Charbel Farhat, Loula Fatima Fezoui

UNITÉ DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cédex
France
Tél. (1) 39 63 55 11

Rapports de Recherche

N° 1322

Programme 7
Calcul Scientifique,
Logiciels Numériques et Ingénierie Assistée
par Ordinateur

STRUCTURED COMPRESSIBLE FLOW COMPUTATIONS ON THE CONNECTION MACHINE

Stéphane LANTERI
Charbel FARHAT
Loula FEZOUI

Novembre 1990

RR-1322

STRUCTURED COMPRESSIBLE FLOW COMPUTATIONS ON THE CONNECTION MACHINE

++Stéphane LANTERI
+Charbel FARHAT , ++Loula FEZOUI

+ Department of Aerospace Engineering
and Center for Space Structures and Controls
University of Colorado
Campus Box 429
BOULDER, COLORADO 80309
++ INRIA Sophia-Antipolis
2004, Route des Lucioles
06560 VALBONNE (FRANCE)

STRUCTURED COMPRESSIBLE FLOW COMPUTATIONS ON THE CONNECTION MACHINE

Stéphane LANTERI, Charbel FARHAT, Loula FEZOUI

ABSTRACT : We present our first experiments on Computational Fluid Dynamics on the Connection Machine CM-2. We are interested in numerical solutions of time-dependent Euler and Navier-Stokes equations using a finite-volume method on structured grids. We would like to obtain reference performances for our approach to massively parallel CFD. These performances are compared to those obtained on a classical supercomputer such as the CRAY-2.

CALCULS D'ECOULEMENTS DE FLUIDES COMPRESSIBLES EN MAILLAGES STRUCTURES SUR LA CONNECTION MACHINE

RESUME : On présente les résultats de nos premières expériences en Mécanique des Fluides Numériques sur la Connection Machine CM-2. On s'intéresse à la résolution numérique des systèmes d'équations d'Euler et de Navier-Stokes en régime instationnaire. On utilise une méthode de volumes finis sur des maillages structurés. Le but de ce travail est d'obtenir des mesures de performance qui servent de référence à notre approche de la MFN sur des calculateurs massivement parallèles. Ces performances sont comparées à celles obtenues sur un supercalculateur classique tel que le CRAY-2 par exemple.

Table des matières

1 Introduction

It is well known that large problems of Computational Fluid Dynamics can be efficiently solved on parallel computers. Different strategies exist, depending on the architecture of the target computer, that permit to decrease the execution time and consequently lead to the solution of larger and larger problems. Among the different classes of parallel machines, we consider here an SIMD one, namely the Connection Machine Model CM-2.

This type of architecture presents several advantages and perhaps the most important one is the facility to program it. Due to its inherent architecture, a Single Instruction Multiple Data machine does not need any synchronisation mechanism between processes; all processors perform the same job at a given time.

The Connection Machine is a massively parallel computer. It is particularly interesting when solving problems that involve intensive computational steps. Computational Fluid Dynamics problems present this characteristic but it is clear that many numerical methods exist to solve a given problem and not all will be efficiently implemented on this machine.

We are interested in the solution of unsteady bi-dimensional flows that can be modelled by Euler or Navier-Stokes equations. In this last case we will solely consider laminar flows. This work represents a first step towards the solution of more general flows such as turbulent ones that can be studied by direct simulation methods using the complete Navier-Stokes equations on very fine grids.

So our first goal is the construction and evaluation of accurate numerical methods to solve non-linear systems of equations describing fluid flows. Several authors have already studied such problems. Jespersen and Levit [4] have developed a finite-difference algorithm to solve the compressible Navier-Stokes equations in two and three dimensions. The essential ingredients of the algorithm are central differencing in space, with explicit and implicit artificial dissipation, and explicit or implicit time-marching. They include comparisons of their code with standard ones that are running on a CRAY X-MP/48. They conclude that for explicit time-stepping, the Connection Machine can outperform a CRAY X-MP while for implicit time-stepping the CM-2 is not quite as fast as the CRAY. Long et al. [5] have made a similar study using a finite-volume, Runge-Kutta time-marching scheme on structured and unstructured grids. They have designed two schemes, one is based on central-differencing augmented by numerical dissipation to maintain stability while the other used an upwind scheme of Roe. Comparisons with a CRAY X-MP show that the unstructured grid code on the Connection Machine is roughly as efficient as the corresponding one on

the vector computer. On the other hand the CM-2 is 15 times faster than the CRAY when working on structured grids. Other comparisons between several computers of different architecture types can be found in the paper of Agarwal [6]. Farhat et al. [7] have solved the full Navier-Stokes equations using the explicit second-order accurate predictor-corrector MacCormack scheme [16] and have achieved a performance of about 1.4 GFLOPS on a 1024×1024 structured grid.

The main ingredients of the present algorithm are the following. For spatial discretization we use a finite volume technique on structured grids based on the M.U.S.C.L. (Monotonic Upwind Scheme for Conservation Laws) method [19] : the convective fluxes are discretized using an upwind scheme while diffusive ones are approximated with the use of the classical centered scheme with the constraint to be consistent with the finite volume formulation. Upwind discretization yields particularly robust schemes that are capable to simulate strong and complex flow conditions. This is achieved through the construction of a numerical flux function that can be viewed as the sum of a centered term and a diffusive one. We consider here two classical formulations, the flux vector splitting of van Leer [13] and the flux vector differencing of Roe [12]. The first one is well suited to the simulation of inviscid flows at all Mach numbers but presents some shortcomings when considering viscous flows [15]. The latter is more appropriate in this last case mainly because it introduces less numerical diffusivity. Finally time integration is accurately obtained via a Runge-Kutta multi-step method with minimal storage requirement.

As it can be seen this general algorithm has been already studied and extensively used; in fact we are mainly interested in how this algorithm can be efficiently implemented on the Connection Machine. Also, we are interested in performance comparison with supercomputers such as the CRAY 2. More precisely, we would like to obtain reference performances for our approach to massively parallel CFD on structured grids. We have also to keep in mind that such discretizations do not permit the simulation of flows in complex geometries. On the other hand finite-element methods on unstructured grids are particularly attractive and often used for such general flows; our experience with these irregular computations while be documented in a forthcomming report.

2 The Connection Machine architecture

In this paragraph we present the main features of the machine and we shall try to emphasize what is particularly interesting for our computations. Further and more precise descriptions of the architecture of the machine can be found in [2] and in the diverse technical reference manuals [1]. The Connection Machine can be defined as a data parallel computing system, that is a system which associates one processor with each data element. In this sense, this computing style exploits the natural computational parallelism inherent in many data intensive problems. Many iterative problems of Computational Fluid Dynamics present this characteristic and the efficiency of the machine on such problems increases with the use of larger and larger sets of data. In the best cases execution time can be reduced in proportion to the number of data elements in the computation.

On the other hand, programming the machine is particularly simple and it can be said that programming effort can be reduced in proportion to the complexity of expressing a naturally parallel statement in a serial manner.

2.1 System organization

The Connection Machine Model CM-2 is an integrated system of hardware and software. The hardware elements of the system include front-end computers that provide the development and execution environments for the system software, a parallel processing unit of 64K processors that execute the data parallel operations, and a high-performance data parallel I/O system (see Figure 1). The system software is based upon the operating system or environment of the front-end computer so that users can program using familiar languages and programming constructs as well as the development tools provided by the front end. Programs have normal sequential control flow, new synchronization structures are not needed.

At the heart of the Connection Machine system is the parallel processing unit which consists of thousands of processors each with thousands of bytes of memory. These processors not only can process the data stored in their memory, but can also be logically interconnected so that information can be exchanged among the processors. All these operations are performed in parallel on all processors.

An important pratical feature of the CM-2 is the support for virtual processors. When the CM-2 is initialized for a run, the number of virtual processors may be specified. If it exceeds the number of available physical processors, then the local memory of each processor is split up into a number of regions equal

the ratio between the number of virtual processors and the number of physical processors (VPR in the sequel). This allows the user to write programs assuming the number of processors that is natural for the application rather than forcing his code to conform to the number of hardware processors. Each hardware processor is made to simulate the appropriate number of virtual processors, as the program issues each parallel instruction, microcode causes it to be executed many times, once for each virtual processor.

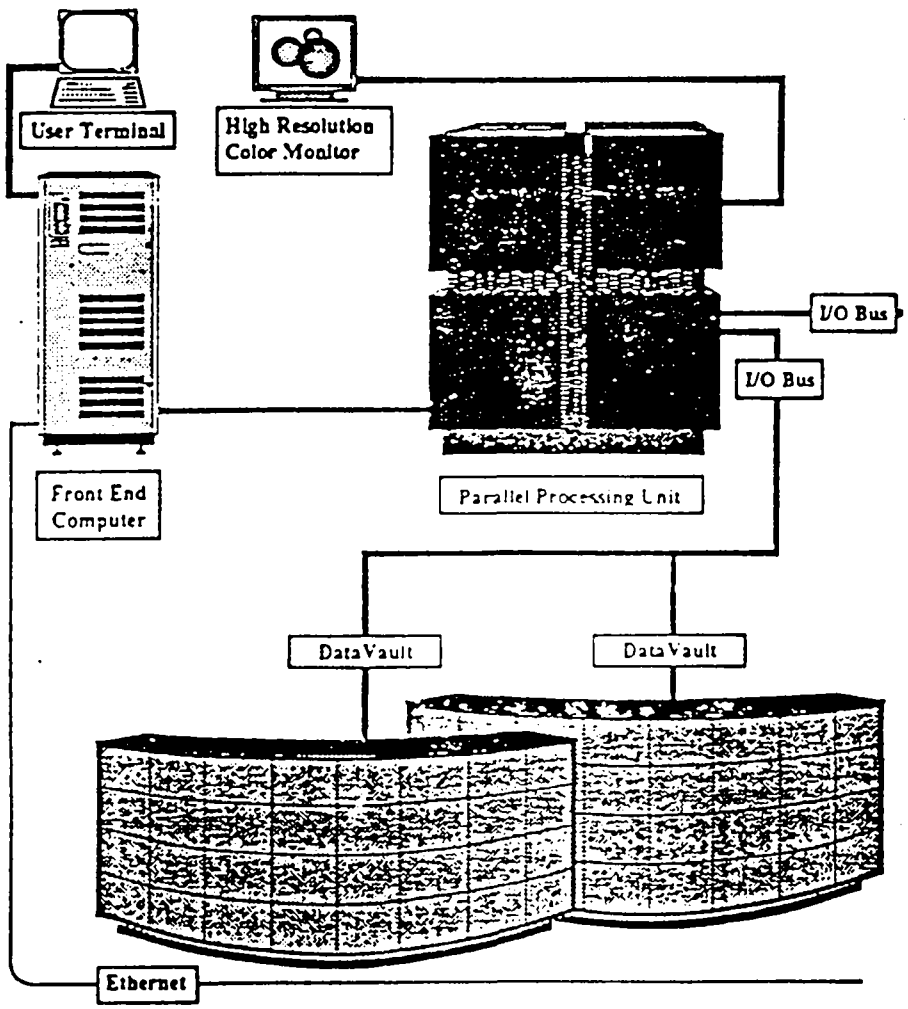

Figure 1 : The Connection Machine system

2.2 Processor architecture

Each elementary data processor consists of (see Figure 2) :

. an arithmetic-logic unit (ALU) and associated latches

. 256 K bits of bit-addressable memory

. four 1-bit hardware flag registers

. optional floating-point accelerator

. router interface

. NEWS grid interface

. direct hypercube interface

. I/O interface

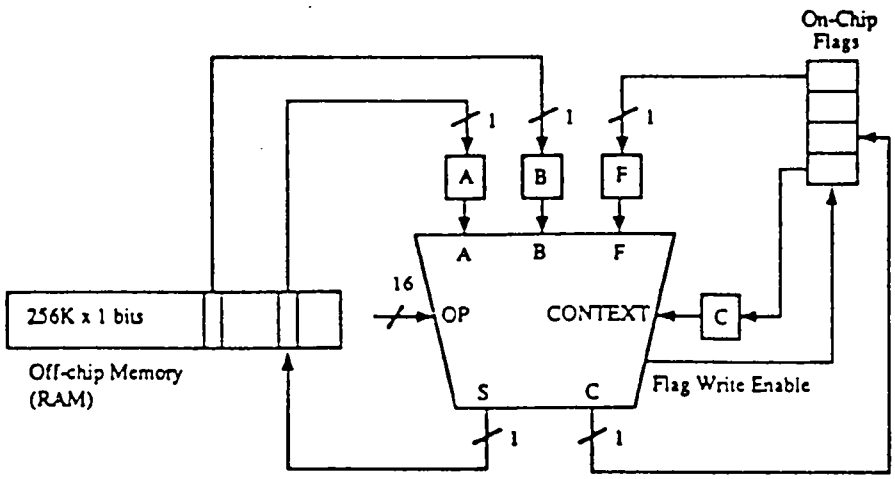

Figure 2 : The Bit-serial data processor

A CM-2 ALU consists of a 3-input, 2-output logic element and associated latches and memory interface (see Figure 3). The basic ALU cycle first reads two data bits from memory and one data bit from a flag. The logic element

then computes two results bits from the three input bits. Finally, one of the two results is stored back into memory and the other result into a flag. An additional feature is that the entire operation is conditional on the value of a third flag. If the flag is zero, then the results for that data processor are not stored after all.

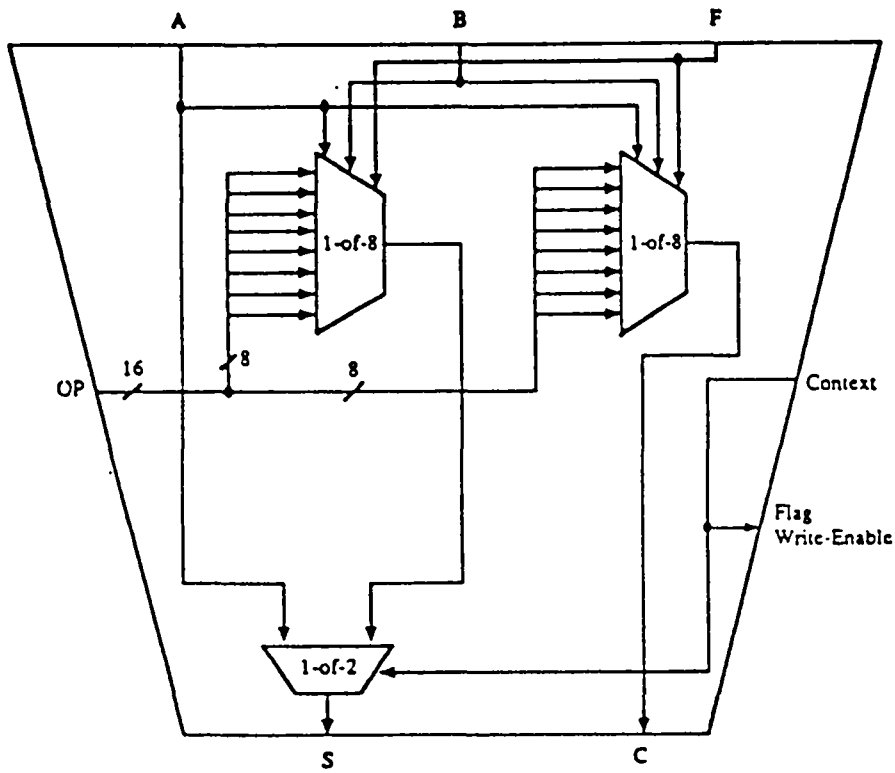

Figure 3 : The Bit-serial ALU

All these data processors are arranged by groups of 16 elements on a CM-2 custom processor chip together with the diverse communication interfaces. A fully configured parallel processing unit contains 64K data processors, and therefore contains 4096 processors chips and two gigabytes of RAM.

In addition to the bit-serial data processor described above, the CM-2 parallel processing unit may be configured with a floating-point accelerator that is closely integrated with the processing unit. There are two possible options for this accelerator : single precision (32-bit) or double precision (64-bit); here we are using the first option. In each case, the rate of floating-point calculations is increased by more than a factor of 20. Taking advantage of this speed increase requires no change in user software. This is an important feature for our computations which are mainly concerned with floating-point manipulations even though the single-precision constraint represent a severe limitation for some numerical methods such as spectral ones. The hardware associated with each of these options consists of two special purpose VLSI chips for each pair of CM-2 processor chips : a memory interface unit and a floating-point execution unit (see Figure 4).

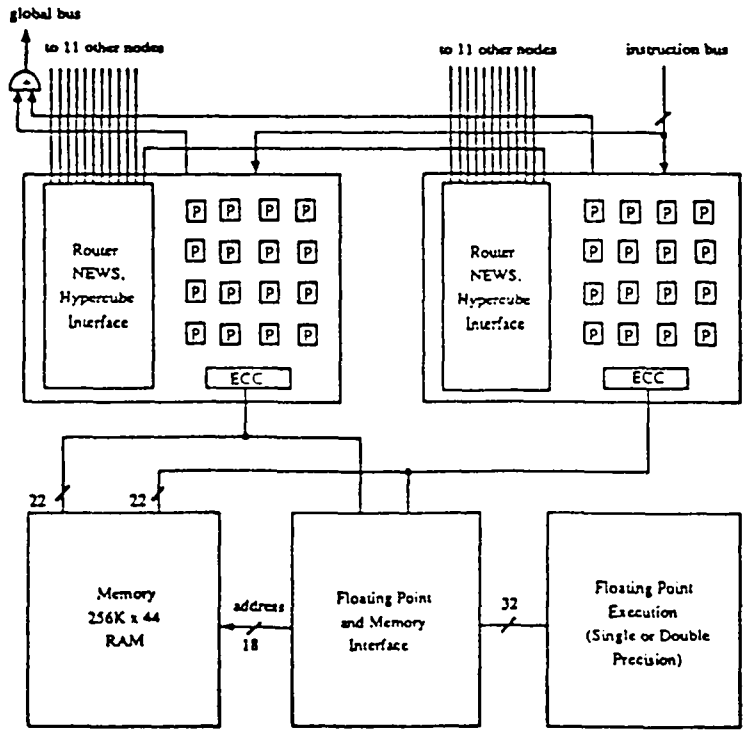

Figure 4 : Two CM2-nodes with memory and floating-point chips

As an example of the operation of the floating-point accelerator, consider the execution of a two-operand 32-bit floating-point instruction such as add or multiply. Execution proceeds in five stages, each stage consists of 32 nanoinstruction

cycles (one cycle for each of the 32 data processors on the two CM-2 processor chips) :

1 The first operand for each of 32 data processors is transferred from memory to the interface chip.

2 The first operand is transferred from the interface chip to the floating-point execution chip.(The floating-point execution chip is capable of storing 32 values of a given precision.) Simultaneously, the second operand is transferred from memory to the interface chip.

3 The second operand is transferred from the floating-point interface chip to the floating-point execution chip, where the operation is performed. At the end of this stage, the floating-point execution chip contains the 32 results.

4 The results are transferred from the floating-point execution chip to the interface chip.

5 The results are transferred from the interface chip to memory.

On the other hand, there exist a pipeline like mechanism so that when the VPR is equal to n, the above process requires only $3n + 2$ stages instead of $5n$ stages.

2.3 Communication mechanisms

The CM-2 provides two forms of communication between the processors (see Figure 5) :

. a general mechanism known as the router which allows any processor to communicate any other processor. Each CM-2 chip contains one router node i which serves the 16 processors on the chip numbered 16i through 16i+15. The router nodes on all the chips are wired together in a 12-dimensional boolean cube and together form the complete router network.

. a more structured and somewhat faster communication mechanism called the NEWS-grid-Each processor is wired to its four nearest neighbors in a two-dimensional rectangular grid. Communication on the NEWS grid is extremely fast and recommended whenever it is possible.

As we shall see in the next sections, we use this last communication mechanism. In fact, this way of communication is well suited to finite difference methods on structured grids and our spatial discretization techniques are closely related to those methods. On the other part, reducing communication costs as much as possible will make it possible to measure the pure computational performance of the machine which represents the main issue of this work.

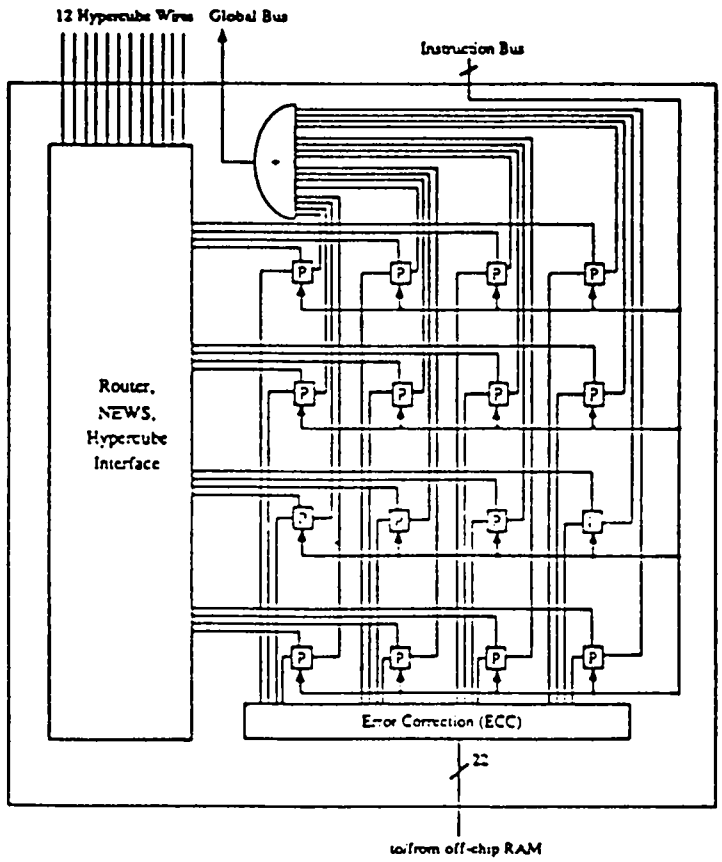

Figure 5 : The CM-2 processor chip and the router

2.4 Benchmarking the CM-2

The CM-2 supports several high level languages :

. CM Fortran which implements the Fortran $8X$ array features directly with no changes to the standard language definition.

. C* (pronounced "see-star") is a parallel extension of the serial C language and is generally used for numeric applications.

. * LISP is an extension of LISP and is commonly used for artificial intelligence and other symbolic processing applications.

On the other part the Connection Machine is provided with a low level instruction set called PARIS (Parallel Instruction Set). It is intended primarly as a base upon which to build higher-level languages for the Connection Machine system. PARIS routines can be called from any of those high-level languages described above but there also exist interfaces with the C and Fortran 77 versions.

We present now performance measurements for the dot product and vector saxpy. We shall emphasize the influence of the VPR and that of the high level language compiler on attainable performances. Considering the first test job we present results obtained with a C* version. The following lines of code give an idea of the new programming style that this language presents.

```
domain void
{
    /* Define parallel variables */
    poly float a;
    poly float b;

};

GRID2-NEWS-MACHINE(VpSet)

main()
{
    /* Define a scalar variable */
    mono float ProdScal;
    [domain void].
```

```
{
a= 2.512;
b= 4.567;
/* Use of reduction operator */
ProdScal += (a*b);
};

}
```

Results are shown in Table 1 where **Nb Proc** represents the number of physical processors used and **Ndim** is the number of virtual processors that are defined.

The same test is now realized using a C/ PARIS version of the code that we present below. Results follow in Table 2.

```
CM_geometry_id_t  Geom;
CM_vp_set_id_t    VpSet;
/* Define memory field identifiers */
CM_field_id_t  a ,b;
/* Define a scalar variable */
float ProdScal;
int Idim[2] , Nx , Ny;

main()

{
CM_init() ;
Idim[0] = Nx;
Idim[1] = Ny;
Geom  = CM_create_geometry(Idim,2);
VpSet  = CM_allocate_vp_set(Geom);
CM_set_vp_set(VpSet);
a  = CM_allocate_heap_field(32);
b  = CM_allocate_heap_field(32);
CM_set_context();
CM_f_move_const_1L(a,2.512,23,8);
CM_f_move_const_1L(b,4.567,23,8);
/* Compute parallel multiplication simultaneously */
/* on all processors */
```

```
CM_f_multiply_2_1L(a,b,23,8);
/* Use of reduction operator */
ProdScal = CM_global_f_add_1L(a,23,8);
CM_deallocate_vp_set(VpSet);
CM_deallocate_geometry(Geom);
}
```

Ndim	NbProc	VPR	MFLOPS
1024K	8K	128	240
2048K	8K	256	435
4096K	16K	256	870

Table 1 : Dot product in C*

Ndim	NbProc	VPR	MFLOPS
2048K	8K	256	425
4096K	16K	256	850
16384K	16K	1024	1135

Table 2 : Dot product in C/PARIS

Several remarks can be made from these results. It is clear that the VPR plays an important role in the resulting efficiency of the two algorithms. Higher performance rates are obtained as the VPR increase but on this point the present version of the C* compiler presents a severe shortcoming. A substancial amount of the processor memory is reserved by the system software so that it hasn't been possible to obtain a result for a VPR equal to 1024 on the 16K machine. On the other hand the above results can be scaled essentially linearly to the 64K processor system. That is if considering the dot product of two vectors of dimension 65536K (VPR=1024) then the performance of 4.5 GFLOPS is attainable on the complete CM-2.

Let's now consider the vector saxpy test problem that is the multiplication of a vector by a scalar number. The previous example involved one step of communication when using the reduction operator, here this is not the case. We

define on each processor memory a vector of dimension 16 and simulteanously multiply all elements by a scalar which is also located on each processor. Table 3 and 4 present the results obtained with the two versions of code and clearly show the superiority over the C* compiler of PARIS.

Ndim	NbProc	VPR	MFLOPS
128K	8K	16	105
128K	16K	8	215

Table 3 : Vector saxpy in C*

Ndim	NbProc	VPR	MFLOPS
128K	8K	16	420
128K	16K	8	840

Table 4 : Vector saxpy in C/PARIS

We also made these tests using the CM Fortran language just for sake of completness. The corresponding results are shown in Tables 1bis and 3bis below.

Ndim	NbProc	VPR	MFLOPS
1024K	8K	128	214
2048K	8K	256	427
4096K	16K	256	854

Table 1bis : Dot product in CM Fortran

Ndim	NbProc	VPR	MFLOPS
128K	8K	16	84
128K	16K	8	141

Table 3bis : Vector saxpy in CM Fortran

3 Computation of inviscid flows

Here we are interested in the solution of the equations of gas dynamics for an inviscid, non-heat conducting fluid : the Euler equations. The two-dimensional and conservative form of these equations are expressed as :

$$\frac{\partial W}{\partial t} + \frac{\partial F(W)}{\partial x} + \frac{\partial G(W)}{\partial y} = 0$$

$$W = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ E \end{pmatrix}$$

$$F(W) = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ (E+p)\,u \end{pmatrix}, \qquad G(W) = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ (E+p)\,v \end{pmatrix}$$

W represents the vector of conservative variables, F and G are the flux vectors in the x and y directions. E is the total energy per unit of volume and p is the pressure given by the state equation :

$$p = (\gamma - 1)\,\rho\varepsilon$$

where $\rho\varepsilon$ represents the internal energie per unit of volume and γ is the ratio of specific heats ($\gamma = 1.4$ for air).
We consider now the following initial and boundary value problem :

$$(1) \qquad \begin{cases} \dfrac{\partial W}{\partial t} + \dfrac{\partial F(W)}{\partial x} + \dfrac{\partial G(W)}{\partial y} = 0 & \left(\vec{X}, t\right) \in \Omega \times \mathbb{R}^+ \\ W\left(\vec{X}, 0\right) = W_0(\vec{X}) & \vec{X} \in \Omega \\ \vec{V}.\vec{\eta} = 0 & \vec{X} \in \partial\Omega \end{cases}$$

The last relation is the classical slip condition and this is the only type of boundary condition we use in this study since we are considering internal fluid flow problems in a closed box.

We are interested in weak solutions of problem (1) that is solutions that can present discontinuities along some lines. Therefore the conservation form of the equations is used in order to well capture such discontinuities (see for example Lomax [9]). In other words this guarantees that correct shock speeds are computed when schock capturing differencing schemes are used.

3.1 Finite volume discretization

We now introduce the following integral form of the problem (1):

$$\int\int_{\Omega} \left(\frac{\partial W}{\partial t} + \frac{\partial F(W)}{\partial x} + \frac{\partial G(W)}{\partial y} \right) \Psi dx dy = 0 \tag{2}$$

where Ψ is some test function. The domain Ω is assumed rectangular and is discretized using a regular grid of equally spaced points. Then for each vertex (i,j) we define a control volume C_{ij} by considering mid-points of segments joining two adjacent points in the x and y directions (see Figure 6).

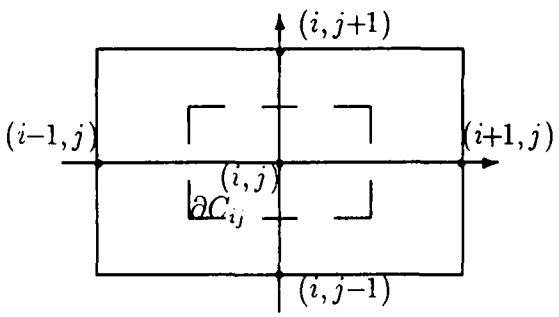

Figure 6 : The control volume on a regular grid

According to this definition we get the following discretization of Ω

$$\Omega_h = \bigcup_{i,j} C_{ij}$$

and now, let Ψ be the characteristic function of the cell C_{ij} defined as :

$$\psi_{ij}(\vec{X}) = \begin{cases} 1 & \text{if } \vec{X} \in C_{ij} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Using (3) and integrating by parts (2) we get the following integral :

$$\int\int_{C_{ij}} \frac{\partial W}{\partial t} dx dy + \int_{\partial C_{ij}} (\eta_x F(W) + \eta_y G(W))\, d\sigma = 0 \tag{4}$$

where $\vec{\eta} = (\eta_x, \eta_y)$ is the outward unit normal at any point of ∂C_{ij}.

3.2 First order approximation

We are concerned here with the spatial discretization of the convective fluxes and more precisely with the boundary integral in (4). First we give the following definitions :

$$\mathcal{F}(W,\vec{\eta}) = \eta_x F(W) + \eta_y G(W)$$

and

$$W_{ij}^n = W(i\Delta x, j\Delta y, n\Delta t)$$

where

$$i \in [0, Nx-1] \quad , \quad j \in [0, Ny-1] \quad \text{and} \quad n \in [0, N]$$

The initial data is given as :

$$W_{ij}^0 = W_0(i\Delta x, j\Delta y)$$

If we suppose that $\frac{\partial W}{\partial t}$ is constant over the cell C_{ij} then (4) becomes :

$$(5) \qquad \begin{aligned} area(C_{ij})\frac{d(W)_{ij}(t)}{dt} \; &+ \sum_{(k,l)\in K(i,j)} \int_{\partial C_{ij}\cap\partial C_{kl}} \mathcal{F}(W,\vec{\eta})d\sigma \\ &+ \int_{\partial C_{ij}\cap\partial\Omega_h} \mathcal{F}(W,\vec{\eta})d\sigma \end{aligned}$$

$K(i,j)$ denotes the set of neighbors of the vertex (i,j). It is clear from (5) that the first integral denotes an internal flux between two adjacent cells and the latter consists in a flux computed at the boundary of the domain Ω_h. We can easily see that :

$$K(i,j) = ((i-1,j),\ (i+1,j),\ (i,j-1),\ (i,j+1))$$

and consequently we have

$$\begin{aligned} \sum_{(k,l)\in K(i,j)} \int_{\partial C_{ij}\cap\partial C_{kl}} \mathcal{F}(W,\vec{\eta})d\sigma \; &= \Phi_{EAST} \; + \Phi_{WEST} \\ &+ \Phi_{SOUTH} + \Phi_{NORTH} \end{aligned}$$

where Φ represents a numerical flux function and is an approximation of the flux between two adjacent cells :

$$\Phi(W_{ij}, W_{kl}, \vec{\eta}_{ij}^{kl}) = \tilde{\mathcal{F}}(W_{ij}, W_{kl}) \int_{\partial C_{ij}\cap\partial C_{kl}} \vec{\eta} d\sigma$$

so that in the case of structured grids the elementary fluxes are computed as :

$$\begin{aligned}
\Phi_{EAST} &= \Phi\left(W_{i+\frac{1}{2}^-,j}\ ,\ W_{i+\frac{1}{2}^+,j}\ ,\vec{\eta}_{EAST}\right)\\
\Phi_{WEST} &= \Phi\left(W_{i-\frac{1}{2}^-,j}\ ,\ W_{i-\frac{1}{2}^+,j}\ ,\vec{\eta}_{WEST}\right)\\
\Phi_{NORTH} &= \Phi\left(W_{i,j+\frac{1}{2}^-}\ ,\ W_{i,j+\frac{1}{2}^+}\ ,\vec{\eta}_{NORTH}\right)\\
\Phi_{SOUTH} &= \Phi\left(W_{i,j-\frac{1}{2}^-}\ ,\ W_{i,j-\frac{1}{2}^+}\ ,\vec{\eta}_{SOUTH}\right)
\end{aligned}$$

where

$$\vec{\eta}_{EAST} = \begin{pmatrix} \int \vec{\eta} d\sigma_{EAST} \\ 0 \end{pmatrix} = -\vec{\eta}_{WEST}$$

$$\vec{\eta}_{NORTH} = \begin{pmatrix} 0 \\ \int \vec{\eta} d\sigma_{NORTH} \end{pmatrix} = -\vec{\eta}_{SOUTH}$$

Consider now a piecewise constant approximation of the unknown W on the cells :

$$\begin{aligned}
W_{i+\frac{1}{2}^-,j} &= W_{i,j} \quad \text{and} \quad W_{i+\frac{1}{2}^+,j} = W_{i+1,j}\\
W_{i-\frac{1}{2}^-,j} &= W_{i-1,j} \quad \text{and} \quad W_{i-\frac{1}{2}^+,j} = W_{i,j}\\
W_{i,j+\frac{1}{2}^-} &= W_{i,j} \quad \text{and} \quad W_{i,j+\frac{1}{2}^+} = W_{i,j+1}\\
W_{i,j-\frac{1}{2}^-} &= W_{i,j-1} \quad \text{and} \quad W_{i,j-\frac{1}{2}^+} = W_{i,j}
\end{aligned}$$

then the resolution of system (5) may be viewed as a collection of local Riemann problems defined at cell interfaces :

$$\begin{cases} W_t + \mathcal{F}(W,\vec{\eta})_\tau = 0 \\ W(\vec{X},t) = \begin{cases} W_L(t) & \text{if } \vec{X} \in C_{ij} \\ W_R(t) & \text{if } \vec{X} \in C_{kl} \end{cases} \end{cases}$$

It is well known that the exact resolution of the Riemann problem may be very costly so it is classical to approximate the solution (as with the approximate Riemann solver of Roe [12]) or just to take into account the direction of waves by splitting the fluxes (as with the flux vector splitting of van Leer [13]).

Before giving more details about such solvers, we remark that according to the finite volume discretization we get the following symmetry property :

$$\Phi(W_L, W_R, \vec{\eta}) = -\Phi(W_R, W_L, -\vec{\eta})$$

so that we just have to compute two fluxes (for instance the EAST and NORTH ones) the others being given by the fluxes computed at neighboring cells (that is the WEST and SOUTH cells).

There is not a unique way to define the numerical flux function Φ; the simplest receipe is to average the fluxes corresponding to the left and right states but this leads to central differencing which is inherently unstable. On the other hand it is well established that robustness of the resulting scheme can be achived using an upwind formula to define the numerical flux function. Following Harten and Al. [19] this may be written as :

$$\Phi(W_L, W_R, \vec{\eta}) = \frac{\mathcal{F}(W_L, \vec{\eta}) + \mathcal{F}(W_R, \vec{\eta})}{2} - d(W_L, W_R, \vec{\eta})$$

where the numerical diffusive term is given by

$$d(W_L, W_R, \vec{\eta}) = \mathcal{Q}(W_L, W_R, \vec{\eta})(W_R - W_L)$$

and $\mathcal{Q}(W_L, W_R, \vec{\eta})$ is the dissipation matrix.

3.2.1 Flux vector splitting

Here we consider the flux vector splitting of van Leer which presents several advantages, among them its simplicity and its robustness in solving critical flows. Before exhibiting the expressions of the different components of the split fluxes, we recall an interesting property of any linear combination of fluxes :

$$\mathcal{F}(W) = \eta_1 F(W) + \eta_2 G(W)$$

Given the reals η_1 et η_2 we define a local rotation on the vector $\vec{V}$ in the following manner :

$$\vec{V} = \begin{pmatrix} u \\ v \end{pmatrix} \longrightarrow \vec{V}_\eta = \begin{pmatrix} u_n = u cos\theta + v sin\theta \\ v_n = -u sin\theta + v cos\theta \end{pmatrix}$$

where we have defined :

$$cos\theta = \frac{\eta_1}{\| \vec{\eta} \|} \qquad sin\theta = \frac{\eta_2}{\| \vec{\eta} \|}$$

Let $\mathcal{R}$ denotes the transformation of $\mathbb{R}^4$ define as :

$$W \rightarrow \mathcal{R}(W) = \begin{pmatrix} \rho \\ \rho \vec{V}_n \\ E \end{pmatrix} = \hat{W}$$

then it is easy to verify that we have the following result :

$$\mathcal{F}(W,\vec{\eta}) = \| \vec{\eta} \| \, (\mathcal{R}^{-1} o \;\; F)(\hat{W})$$

which shows that it's possible to use solely the component F to compute $\mathcal{F}$ and this yield to a noticeable economy of computing time (for more details on this property see Fezoui and Steve [11]). The classical splitting of van Leer [13] is defined by

$$F(\hat{W}) = f^+(\hat{W}) + f^-(\hat{W})$$

where the expressions of the split fluxes are obtained with the use of the local Mach number $M_n = \frac{u_n}{c}$.

$$f^+(\hat{W}) = \begin{cases} F(\hat{W}) & \text{if } M_n \geq 1 \\ \begin{cases} \frac{\rho c}{4}\left(\frac{u_n}{c}+1\right)^2 = f_1^+ \\ \frac{f_1^+}{\gamma}\left((\gamma-1)u_n + 2c\right) \\ f_1^+\left(\frac{((\gamma-1)u_n+2c)^2}{2(\gamma^2-1)} + \frac{v_n^2}{2}\right) \end{cases} & \text{if } \mid M_n \mid < 1 \\ 0 & \text{if } M_n \leq -1 \end{cases}$$

Getting these results into the expression of the numerical flux function Φ completes the definition of the spatial discretization :

$$\Phi(W_L, W_R, \vec{\eta}) = \| \vec{\eta} \| \, \mathcal{R}^{-1}\left(f^+(\hat{W}_L) + f^-(\hat{W}_R)\right)$$

3.2.2 Flux vector differencing

We now consider the approximate Riemann solver of Roe which is defined by the expressions :

$$(6) \qquad \Phi\,(W_L, W_R, \vec{\eta}) = \frac{\mathcal{F}\,(W_L, \vec{\eta}) + \mathcal{F}\,(W_R, \vec{\eta})}{2} - d\,(W_L, W_R, \vec{\eta}) \;,$$

where $d\,(W_L, W_R, \vec{\eta})$ is a numerical diffusivity term which is defined as :

$$(7) \qquad d\,(W_L, W_R, \vec{\eta}) = \mid \mathcal{R}\,(W_L, W_R, \vec{\eta}) \mid \frac{(W_R - W_L)}{2} \;,$$

$\mathcal{R}(W_L, W_R, \vec{\eta})$ is referred as Roe's matrix and has some particular properties. Among them we recall about the following one :

$$\mathcal{R}(W_L, W_R, \vec{\eta})(W_L - W_R) = \mathcal{F}(W_L, \vec{\eta}) - \mathcal{F}(W_R, \vec{\eta}) \tag{8}$$

wich yields several remarks about the complete scheme. First, it can be seen that the scheme solves exactly steady contact discontinuities. More precisely if the right and left states W_L and W_R define a contact discontinuity then Rankine-Hugoniot relations are written as $\mathcal{F}(W_L, \vec{\eta}) = \mathcal{F}(W_R, \vec{\eta})$, and the expressions (6)-(7)-(8) imply that $d^R(W_L, W_R, \vec{\eta}) = 0$ and $\Phi(W_L, W_R, \vec{\eta}) = \mathcal{F}(W_L, \vec{\eta})$.

This scheme is associated with a small amount of numerical diffusivity compare to the flux vector splitting of van Leer. This represents a substancial advantage when simulating viscous flows where the physical viscosity is dominating. On the other part the property (8) has another consequence on the overall scheme yielding the two following simplified expressions :

$$\Phi(W_L, W_R, \vec{\eta}) = \mathcal{F}(W_R, \vec{\eta}) - \mathcal{R}^+(W_L, W_R, \vec{\eta})(W_R - W_L) \ ,$$

or

$$\Phi(W_L, W_R, \vec{\eta}) = \mathcal{F}(W_L, \vec{\eta}) + \mathcal{R}^-(W_L, W_R, \vec{\eta})(W_R - W_L)$$

which contributes to decreasing the computing time needed to solve a Riemann problem.

The matrix $\mathcal{R}(W_L, W_R, \vec{\eta})$ which verifies the property (8) is not unique, nevertheless Roe has proposed an expression of $\mathcal{R}$ using the Jacobian matrix of the flux $\mathcal{F}(W, \vec{\eta})$:

$$\mathcal{R}(W_L, W_R, \vec{\eta}) = \mathcal{A}\left(\tilde{W}, \vec{\eta}\right) \ ,$$

where $\tilde{W}$ is the mean value of Roe of the states W_L and W_R, defined as

$$W_L = \begin{pmatrix} \rho_1 \\ \rho_1 u_1 \\ \rho_1 v_1 \\ E_1 \end{pmatrix} \ , \quad W_R = \begin{pmatrix} \rho_2 \\ \rho_2 u_2 \\ \rho_2 v_2 \\ E_2 \end{pmatrix} \ , \quad \tilde{W} = \begin{pmatrix} \tilde{\rho} \\ \tilde{\rho}\tilde{u} \\ \tilde{\rho}\tilde{v} \\ \tilde{E} \end{pmatrix} \ ,$$

with

$$\tilde{\rho} = \frac{\sqrt{\rho_1}\rho_1 + \sqrt{\rho_2}\rho_2}{\sqrt{\rho_1} + \sqrt{\rho_2}} \ ,$$

$$\tilde{u} = \frac{\sqrt{\rho_1}u_1 + \sqrt{\rho_2}u_2}{\sqrt{\rho_1} + \sqrt{\rho_2}} \ ,$$

$$\tilde{v} = \frac{\sqrt{\rho_1}v_1 + \sqrt{\rho_2}v_2}{\sqrt{\rho_1} + \sqrt{\rho_2}} ,$$
$$\tilde{H} = \frac{\sqrt{\rho_1}H_1 + \sqrt{\rho_2}H_2}{\sqrt{\rho_1} + \sqrt{\rho_2}} ,$$

$H = \frac{\gamma p}{(\gamma - 1)\rho} + \frac{u^2 + v^2}{2}$ is the total enthalpy per unit of volume.

3.3 Second order extension

The scheme so far described is first order accurate in space and it's main shortcoming is that it suffers from too much diffusivity (in other words the upwind approximation introduces an inportant amount of numerical diffusivity). Second order precision can be obtained using Piecewise Linear Interpolation of left and right states at cell interfaces [14]. Using Taylor's expansion of the function $W(\vec{X}, t)$ we can deduce :

$$(9) \qquad \begin{aligned} W_{i+\frac{1}{2}^-,j} &= W_{i,j} + \frac{1}{2}\frac{\partial W}{\partial x}\Big|_{i,j} \\ W_{i,j+\frac{1}{2}^-} &= W_{i,j} + \frac{1}{2}\frac{\partial W}{\partial y}\Big|_{i,j} \end{aligned}$$

and it can be seen that we now need to calculate the nodal gradients $\frac{\partial W}{\partial x}|_{i,j}$ and $\frac{\partial W}{\partial y}|_{i,j}$. This is done by combining centered and fully upwind differences of adjacent states :

$$(10) \qquad \begin{aligned} \frac{\partial W}{\partial x}\Big|_{i,j} &= (1-\beta)(W_{i+1,j} - W_{i,j}) + \beta(W_{i,j} - W_{i-1,j}) \\ \frac{\partial W}{\partial y}\Big|_{i,j} &= (1-\beta)(W_{i,j+1} - W_{i,j}) + \beta(W_{i,j} - W_{i,j-1}) \end{aligned}$$

here β denotes an upwind parameter and classical schemes are obtained for the following values (see Desideri [10]) :

$$(11) \qquad \begin{aligned} &\beta = 0 \Rightarrow \text{centered,} \\ &\beta = 1 \Rightarrow \text{fully upwind,} \\ &\beta = \tfrac{1}{2} \Rightarrow \text{Fromm's Scheme,} \\ &\beta = \tfrac{1}{3} \Rightarrow \text{third order accurate in the linear case.} \end{aligned}$$

Recall that when we say "centered" this mean "centered at cell interface" that is at $(i+\frac{1}{2}, j)$ or $(i, j+\frac{1}{2})$. It is well known that the solutions of nonlinear problems by higher-order numerical schemes suffer from dispersive errors visible

by the presence of "riplles" in the plots of the unknowns and particularly near steep gradients. This can be overcomed with the use of flux-correctors or limiters [14]. We won't consider such operators here just for reasons of simplicity but their implementation is straightforward.

3.4 Boundary conditions

Our problem involve the boundary condition $\vec{V}.\vec{\eta} = 0$. It's possible to take into account this condition in the weak formulation, in fact it's easy to verify that :

$$\vec{V}.\vec{\eta} = 0 \Rightarrow \int_{\partial C_{i,j} \cap \partial\Omega_k} \mathcal{F}(W, \vec{\eta}) d\sigma = p \begin{pmatrix} 0 \\ \eta_x \\ \eta_y \\ 0 \end{pmatrix}$$

which define the boundary flux.

3.5 Time integration

Once the spatial discretization is accomplished we obtain the following system of ordinary differential equations :

$$\frac{dW}{dt} + \psi(W) = 0$$

Time integration is then accurately obtained using an explicit Runge-Kutta method, here we use a 3 steps algorithm :

$$\begin{cases} W^{(0)} = W^n \\ W^{(1)} = W^{(0)} - \dfrac{\Delta t}{3}\psi(W^{(0)}) \\ W^{(2)} = W^{(0)} - \dfrac{\Delta t}{2}\psi(W^{(1)}) \\ W^{(3)} = W^{(0)} - \Delta t\psi(W^{(2)}) \\ W^{n+1} = W^{(3)} \end{cases}$$

This scheme is often referred as a low-storage Runge-Kutta method as we only use the solution at step $\alpha - 1$ to compute the one at step α. It is third order accurate in the linear case but only second-order in the present.

3.6 General algorithm and numerical results

We first present the general algorithm used to find unsteady solutions of Euler's equations.

loop

/* Loop on the sub-steps of Runge-Kutta */
loop

Get and store EAST and NORTH states
Compute and store nodal gradients
Compute and store EAST and NORTH fluxes
Compute new solution

until *nb-sub-steps=max-sub-steps*

until *t=tmax*

In designing our code we have fixed ourselves two main goals :

1 Minimize memory storage : recall that we are interested in direct simulation methods which are characterized by the definition of large data sets, so we would like to be able to use the highest possible VP ratio.

2 Minimize communication costs : we also want to obtain reference performances for our applications on the CM-2, that is measures for the pure computing capabilities of the machine.

Let us give some precision on how these issues have been inserted in the above algorithm. Each data-processor is associated with one computational point or vertex of the spatial discretization and the corresponding memory has been used as follows :

- Variables related to the cell geometry.
- Initial state of Runge-Kutta method $\rho 0$, $\rho u 0$, $\rho v 0$ and $e 0$.
- ρ, u, v, e and p of the vertex.
- ρ, u, v, e and p of the EAST neighbor.
- ρ, u, v, e and p of the NORTH neighbor.
- $\vec{\nabla}\rho$, $\vec{\nabla}u$, $\vec{\nabla}v$ and $\vec{\nabla}p$ of the vertex.
- The four components of the EAST flux.

- The four components of the NORTH flux.

This set of global variables represents 46 real numbers, each stored on 32 bits. Taking into account temporary storage for local variables we get a total of about 64 floating-point numbers for a cost of 2048 bits. Then referring to the 256 Kbits of available memory per processor we can deduce the maximum VP ratio attainable which is equal to 128. This leads to 1 million computational points on the 8K machine and eight times more on the complete CM-2.

On the other part, the main loop involves three communication steps. First, we get EAST and NORTH neighbor's states using several calls to the appropriate NEWS routine for each direction. In practice all processors communicate simultaneously with their EAST or NORTH neighbor and this is done really efficiently with the NEWS grid package. Computation of the nodal gradients also needs to communicate with NORTH and SOUTH neighbors, and here it's also possible to combine elementary operations (add, subtract or multiply) and communication within the same routine (recall that we don't need to store these neigbors). Finally the last step of communication is introduced when accumulating the elementary fluxes as we used the symmetry property for WEST and SOUTH boundary exchanges.

This algorithm is now tested on the following problem that can give a simplified representation of the strong acoustic waves that may occur during an engine knock. The flow domain is rectangular of dimensions $[0,1]\times[0,1]$ with adiabatic slipping walls (see Figure 7). Different initial conditions are used in a subdomain $\mathcal{D}_1$ of dimensions 0.4×0.4 and in the remaining part $\mathcal{D}_0$ of $\mathcal{D}$.

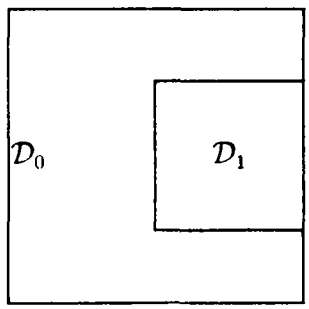

Figure 7 : Flow domain for test problem

The initial conditions are the following :

$$\begin{cases} \rho_0 = 1 \quad u_0 = v_0 = 0 \quad p_0 = 0.375 \\ \\ \rho_1 = 1 \quad u_1 = v_1 = 0 \qquad p_1 = 1 \end{cases}$$

As the flow evolves, a travelling wave front develops from region $\mathcal{D}_1$ and the complexity of the phenomena increases when the different waves join the walls. The solution at t=0.25 for a mesh of 64×128 points is shown on Figure 8 that represents iso-pressure lines in the domain $\mathcal{D}$.

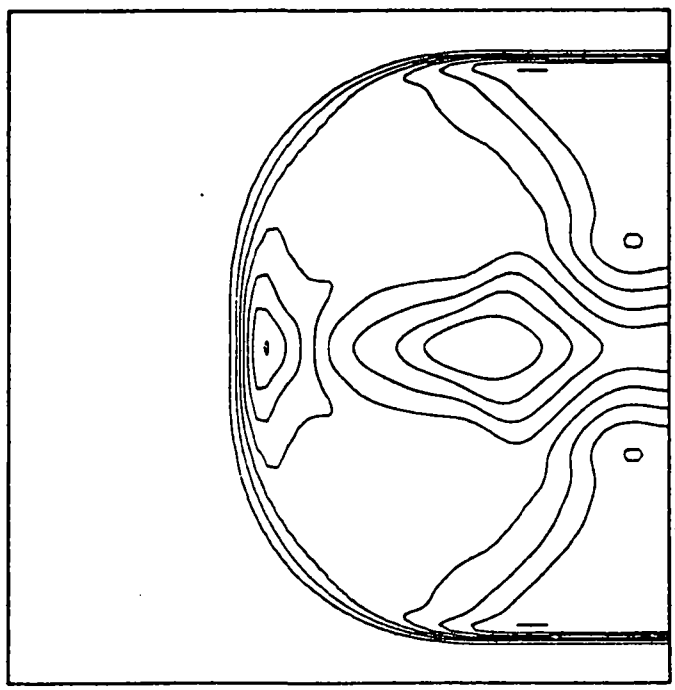

Figure 8 : Iso-pressure lines in $\mathcal{D}$

We now give numerical results obtained with several versions of the above algorithm depending on the language or even the machine used. Firstly we compare CPU-times obtained with the C* version (Table 5) and the C/PARIS one (Table 6) in order to show the power of the latter. The numerical flux function used is the van Leer's one. We also give the number of time steps (*Iter*) that are necessary to reach time 0.25.

$Nx \times Ny$	$NbProc$	VPR	$Iter$	$CPU(sec)$
64×128	$8k$	1	53	44.2
128×128	$8k$	2	59	96.5
128×128	$16k$	1	59	48.9

Table 5 : C* version

$Nx \times Ny$	$NbProc$	VPR	$Iter$	$CPU(sec)$
64×128	$8k$	1	53	3.2
128×128	$8k$	2	59	6.2
128×128	$16k$	1	59	3.8

Table 6 : C/PARIS version

In the following we shall consider solely the C/PARIS version as it is clear that it gives the best results but before going on we must precise that a new C* compiler is under developpment which is expected to yield a more efficient code.

We now present several results that are associated with two types of comparison.

First we want to emphasize the influence of the choice of the numerical flux function. Let's consider the case of van Leer's flux vector splitting. The amount of work done to solve one Riemann problem is not the same for each processor as it clearly depends on the value of the local Mach number. In practice some processors are doing some work while others are idle, then when the previous ones end their work the others make the coresponding computations (see the different cases in the definition of the split fluxes in paragraph (3.2.1)). On the other part, in the case of Roe's flux vector differencing all processors are doing the same amount of work. This will give an idea of how the CM-2 system handles a if statement as micro-instructions have to be broadcasted to a subset of all the processors.

Second we would like to give some results for the comparison of the CM-2 with another machine commonly used in such studies, namely the CRAY-2. On this last point we must precise that the two algorithms used are not strictly the same as one computer is of parallel type and the other one belongs to the vector class.

In table 7 below we give results obtained on the 8K processors machine with the van Leer's flux splitting. Each processor compute its number of floating point operations during one time step then all the results are added to deduce the MFLOPS rate. Table 8 lists the corresponding results obtained on the CRAY-2 mono-processor.

$Nx \times Ny$	VPR	$Iter$	CPU	$MFLOPS$
64 × 128	1	53	3.2	120
128 × 128	2	59	6.2	137
256 × 128	4	103	19.8	151
256 × 256	8	121	41.3	170
512 × 256	16	218	142	178
512 × 512	32	250	310	187
1024 × 512	64	469	1142	191

Table 7 : Results on the CM-2 with van Leer's flux vector splitting

$Nx \times Ny$	$Iter$	CPU	$MFLOPS$
64 × 128	53	4.0	157
128 × 128	59	8.9	158
256 × 128	103	31.1	159
256 × 256	121	71.4	162
512 × 256	218	256	163
512 × 512	250	584	164
1024 × 512	469	2214	164

Table 8 : Results on the CRAY-2 with van Leer's flux vector splitting

Before going on we show two computations using the 16K processors machine. The first one gives a measure at VPR=2 that can be compared with the one obtained for the same VPR but with 2 times less points on a 8K machine. The second one consists in letting the number of points fixed and dividing the VPR by a factor 2. Using these two results we shall deduce the way of obtaining theoretical results on the complete configuration. More precisely we want to know which of all these results can be linearly scaled to get the highest performance. McBryan [3] has shown that all results on subcubes of the CM scale essentially linearly to the 64K processor system provided that only nearest-neighbor communication is involved. So looking at Table 9 we see that the two

strategies yield different conclusions. If we assume that we use the complete machine with 64K processors and solve Euler's equations on a grid of 1024*2048 computational points (so that VPR=32), which is easily possible in regard to the memory storage we need, then the performance will reach 1.5 GFLOPS. On the other part the second strategy is less exploitable as the CPU time dosen't scale linearly.

$Nx \times Ny$	VPR	$Iter$	CPU	$MFLOPS$
128 × 256	2	107	11.4	273
512 × 512	16	250	166	351

Table 9 : Results on the CM-2 with 16K processors

We now consider the same computations with Roe's flux vector differencing. Results presented in Table 10 show that CPU times have slightly increased but the number of floating point operations involved is greater so that the MFLOPS rates are higher.

$Nx \times Ny$	VPR	$Iter$	CPU	$MFLOPS$
64 × 128	1	53	3.45	154
128 × 128	2	59	6.65	177
256 × 128	4	103	21.4	194
256 × 256	8	121	46	212
512 × 256	16	218	158	221
512 × 512	32	250	347	232
1024 × 512	64	469	1274	237

Table 10 : Results on the CM-2 with Roe's flux vector differencing

Using the complete configuration under the same conditions as expressed previously will theoretically yield a performance of 1.85 GFLOPS.

$Nx \times Ny$	$Iter$	CPU	$MFLOPS$
64 × 128	53	4.7	132
128 × 128	59	10.6	132
256 × 128	103	36.2	134
256 × 256	121	84.6	135
512 × 256	218	301	137
512 × 512	250	697	137
1024 × 512	469	2604	137

Table 11 : Results on the CRAY-2 with Roe's flux vector differencing

From tables 7 and 8 we can see that the Connection Machine outperforms the CRAY (mono-processor) for all VPR. At VPR=32 the CM is roughly twice faster than the CRAY. The difference is less noticeable when comparing CPU times from tables 10 and 11. However MFLOPS rates on the CM are always increasing with the VPR while this is not the case for the CRAY where it seems that the maximum performance has been attained.

On the other hand, it would be of interest to give some precisions about the communication time versus the computation one. We choose Roe's flux vector differencing as the number of floating point operations is the same for all processors. We first give the details of the floating point operations (per computational point) that involve the main steps of the algorithm described previously. We must however indicate that the number given for the last line is rigorously correct for internal points. In fact computational points on the boundary are treated slightly differently to take into account the boundary condition and the corresponding number decreases to 32.

(12)

Computation of the nodal gradients :	40
Computation of the EAST and NORTH fluxes :	336
Computation of the new solution :	34

Let's now give the number of communications in each step. The same remark must be done for the last step where the boundary points only communicate 4 values.

(13)

Get EAST and NORTH states :	8
Computation of the nodal gradients :	8
Computation of the EAST and NORTH fluxes :	8
Computation of the new solution :	8

Let's consider an internal computational point, then it is associated with 32 communications for 410 floating point operations (for one time step) so that communications correspond to about 8% of all the work done. In practice we could compare this last number to the percentage of CPU time attached to communication steps. In fact this is not really reasonable as times to compute one floating point operation and to process one communication are no longer comparable. On the other hand, we can get more information by evaluating the computational efficiency of the algorithm which is given by the relation :

$$Ec = \frac{T_{computation}}{T_{execution}} = \frac{T_{computation}}{T_{computation} + T_{communication}}$$

In table 12 below we have listed the values of Ec obtained for the previous computations. Let's recall that some communication steps are combined

with computation ones within the same routine call. Using such instructions contribute to noticeably increase the efficiency of the algorithm particularly at high VPR. However it becomes difficult to separate the communication time from the computation one. Here we choose to place ourselves in the worst situation considering each call to these instructions as a communication step.

$Nx \times Ny$	$T_{execution}$	$T_{computation}$	$Ec\%$
64 × 128	3.45	2.74	79.7
128 × 128	6.65	5.22	78.4
256 × 128	21.4	18.3	85.5
256 × 256	46	40	86.9
512 × 256	158	140	88.6
512 × 512	347	307	88.4
1024 × 512	1274	1162	91.2

Table 12 : Computational efficiency Ec for Roe's flux vector differencing

The above results clearly show that communication costs become less significant as the VPR increases or as the number of computational points is greater.

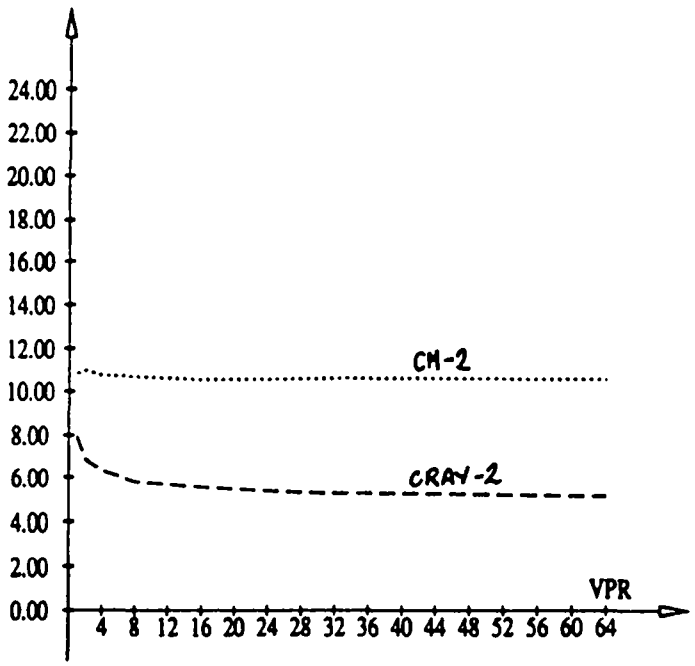

Figure 9 : CPU cost for one time step per point versus VPR

As a last result, we compare the CM-2 and the CRAY-2 with the evolution of the CPU cost for one time step per computational point. On Figure 9 above

VPR=1 corresponds to a grid of 64×128 points and time CPU on the y-axis are given in μsec. The dashed curve is obtained for the CM-2 with 8K physical processors. As the VPR increases and tends to its maximum the ratio between the two machines became constant and we can see that the CM-2 is twice faster than the CRAY-2. On the complete configuration and for the highest VPR this ratio should be of 16.

4 Computation of viscous flows

The complete set of Navier-Stokes equations constitute the more general model for Computational Fluid Dynamics [8], in particular it includes direct modeling of turbulence. In practice this is possible with the use of very fine grids to insure the correct capturing of small scale phenomena. When this becomes too costly in regard to the material capabilities we resort to turbulence modelling by defining mean values of turbulent quantities. But this leads to new unknowns and it is then necessary to introduce new equations (turbulence models) to close the problem. Here we consider the first approach where the general equations are expressed as follows :

$$\frac{\partial W}{\partial t} + \frac{\partial F(W)}{\partial x} + \frac{\partial G(W)}{\partial y} = \frac{1}{Re}\left(\frac{\partial R(W)}{\partial x} + \frac{\partial S(W)}{\partial y}\right) \tag{14}$$

$$W = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ E \end{pmatrix}$$

with the same definitions as in the previous part for the convective fluxes. The diffusive fluxes $R(W)$ and $S(W)$ are given by :

$$R(W) = \begin{pmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ u\tau_{xx} + v\tau_{xy} + \dfrac{\gamma k}{Pr}\dfrac{\partial \varepsilon}{\partial x} \end{pmatrix}, \qquad S(W) = \begin{pmatrix} 0 \\ \tau_{xy} \\ \tau_{yy} \\ u\tau_{xy} + v\tau_{yy} + \dfrac{\gamma k}{Pr}\dfrac{\partial \varepsilon}{\partial y} \end{pmatrix}.$$

where ε denotes the specific internal energy related to the other physical quantities by the relation :

$$\varepsilon = \frac{E}{\rho} - \frac{1}{2}(u^2 + v^2) = C_v T ,$$

T is the temperature. Assuming Stokes's hypothesis, the Cauchy stress tensor components can be expressed in the following manner :

$$\tau_{xx} = \frac{2}{3}\mu\left(2\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y}\right),$$

$$\tau_{yy} = \frac{2}{3}\mu\left(2\frac{\partial v}{\partial y} - \frac{\partial u}{\partial x}\right),$$

$$\tau_{xy} = \mu \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right) .$$

μ is the coefficient of viscosity and k is the coefficient of thermal conductivity wich are generally functions of the temperature (using Suntherland's law). Here we will consider them as constants. Two other numbers appear in the above equations; the Reynols number which measures the importance of convection terms over diffusion ones :

$$Re = \frac{\rho_0 U_0 L_0}{\mu_0} ,$$

where ρ_0, U_0, L_0 et μ_0 respectively denote characteristic density, velocity, length and diffusivity of the flow under consideration. The Prandtl number just involves physical quantities of the fluid :

$$Pr = \frac{\mu_0 C_p}{k_0} .$$

The initial and boundary value problem (1) is slightly modified to take into account for the diffusive fluxes :

$$(15) \begin{cases} \dfrac{\partial W}{\partial t} + \dfrac{\partial F(W)}{\partial x} + \dfrac{\partial G(W)}{\partial y} = \dfrac{1}{Re} \left(\dfrac{\partial R(W)}{\partial x} + \dfrac{\partial S(W)}{\partial y} \right) & \left(\vec{X}, t\right) \in \Omega \times \mathbb{R}^+ \\ W\left(\vec{X}, 0\right) = W_0(\vec{X}) & \vec{X} \in \Omega \\ \vec{V} = \vec{V}_{wall} & \vec{X} \in \partial\Omega \end{cases}$$

where the last relation represents the no-slip condition at wall boundaries. Again we consider internal flows in a rectangular domain.

4.1 Finite volume discretization

Week solutions of problem (15) can be obtained with the use of the following integral form :

$$(16) \quad \int\int_\Omega \left(\frac{\partial W}{\partial t} + \frac{\partial F(W)}{\partial x} + \frac{\partial G(W)}{\partial y} - \frac{1}{Re} \left(\frac{\partial R(W)}{\partial x} + \frac{\partial S(W)}{\partial y} \right) \right) \Psi dx dy = 0$$

Spatial discretization of Ω is achieved as previously and we now want to define discrete diffusive fluxes that are consistent with the finite volume formulation of the convective ones. The control volume and test functions are defined in the same manner and integration by parts yields the following integral :

$$\begin{aligned} \iint_{C_{ij}} \frac{\partial W}{\partial t} dx dy \; &+ \int_{\partial C_{ij}} (\eta_x F(W) + \eta_y G(W))\, d\sigma \\ &= \frac{1}{Re} \int_{\partial C_{ij}} (\eta_x R(W) + \eta_y S(W))\, d\sigma \end{aligned} \tag{17}$$

where $\vec{\eta} = (\eta_x, \eta_y)$ is the outward unit normal at any point of ∂C_{ij}.

4.2 Spatial discretization

In this part we define the spatial discretization of the diffusive fluxes, the convective ones being computed with te method describe in paragraph (3.2). In the finite volume approach, the numerical fluxes are evaluated at cell interfaces and the dependent variables are defined at centroids. So let's concentrate on evaluating $R_{i+\frac{1}{2},j}$ that is the viscous flux between cell $C_{i,j}$ and cell $C_{i+1,j}$. It is clear that in order to construct this flux we have to compute several first-derivative terms (u_x, u_y, ε_x ....) at location $i + \frac{1}{2}, j$. Once these are evaluated using appropriate numerical approximations, the remaining steps are computed by algebraic evaluation of the terms of $R(W)$. Many authors have proposed finite volume discretization of the Navier-Stokes equations, among them we choose to implement the method of Chakravarthy and Al. [17]. A similar work can be found in Liou [18] with the use of the Roe's flux vector differencing scheme for the discretization of convective fluxes. Consider the different cells shown by the solid lines in Figure 10 and let's define the auxiliary cell limited by the dashed lines. Its four corners are denoted by NW, SW, SE and NE and the approximate mid-points of the sides are shown as N, W, S, and E. The x and y coordinates of the vertices of this auxiliary cell can be computed by interpolation from the vertices of the original cells. Let A_{aux} denote the aera of the auxiliary cell.

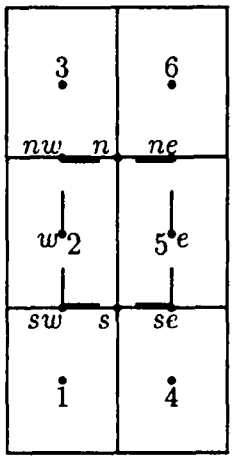

Figure 10 : Computational cells for viscous fluxes

Let's assume that values of u, v and ϵ that are evaluated at points N, W, S and E are also constant along the respective sides. We now make use of Gauss's integral formula applied to the auxiliary cell to obtain approximations to the required derivatives. These approximations denote average values of the first derivatives and can thus be assigned to the centroid of the auxiliary cell :

$$
(18) \qquad \begin{cases} (u_x)_{i+\frac{1}{2},j} \approx \dfrac{1}{A_{aux}} (u_E dy_E + u_N dy_N \\ \qquad\qquad + u_W dy_W + u_S dy_S) \\ (u_y)_{i+\frac{1}{2},j} \approx -\dfrac{1}{A_{aux}} (u_E dx_E + u_N dx_N \\ \qquad\qquad + u_W dx_W + u_S dx_S) \end{cases}
$$

and the approximations to the other derivatives are obtained by substituting the corresponding dependent variable in (18). The metric quantities are defined as follow :

$$
\begin{aligned} dx_E &= x_{NE} - x_{SE} , & dy_E &= y_{NE} - y_{SE} \\ dx_N &= x_{NW} - x_{NE} , & dy_N &= y_{NW} - y_{NE} \\ dx_W &= x_{SW} - x_{NW} , & dy_W &= y_{SW} - y_{NW} \\ dx_S &= x_{SE} - x_{SW} , & dy_S &= y_{SE} - y_{SW} \end{aligned}
$$

In the above expressions, values at points W and E are respectively associated to the computational points i,j and $i+1,j$ but we have not yet defined those at points N and S. This is simply done by taking the average values of the four neighbours surrounding these two points :

$$u_N \approx \frac{1}{4}(u_2+u_3+u_5+u_6)$$

$$u_S \approx \frac{1}{4}(u_2+u_5+u_1+u_4)$$

Turning back to our finite volume discretization, we finally obtain the following equation four each cell C_{ij} :

$$\begin{aligned} area(C_{ij})\frac{\dot{d}(W)_{ij}(t)}{dt} &+ \sum_{(k,l)\in K(i,j)} \int_{\partial C_{ij}\cap\partial C_{kl}} \mathcal{F}(W,\vec{\eta})d\sigma \\ &= \frac{1}{Re}\sum_{(k,l)\in K(i,j)} \int_{\partial C_{ij}\cap\partial C_{kl}} \mathcal{V}(W,\vec{\eta})d\sigma \end{aligned} \tag{19}$$

$K(i,j)$ denotes the set of neighbours of the vertex (i,j) and we have noted

$$\mathcal{V}(W,\vec{\eta}) = \eta_x R(W) + \eta_y S(W)$$

As previously, we define internal viscous fluxes at the different cell interfaces :

$$\begin{aligned} \sum_{(k,l)\in K(i,j)} \int_{\partial C_{ij}\cap\partial C_{kl}} \mathcal{V}(W,\vec{\eta})d\sigma &= \Phi^{vis}_{EST} + \Phi^{vis}_{WEST} \\ &+\Phi^{vis}_{SOUTH} + \Phi^{vis}_{NORTH} \end{aligned}$$

where Φ represents a numerical flux function and is an approximation of the diffusive flux between two adjacent cells. As a consequence of the finite volume formulation, the symmetry property is yet verified

$$\Phi^{vis}(W_L,W_R,\vec{\eta}) = -\Phi^{vis}(W_R,W_L,-\vec{\eta})$$

On the other part, the normal components deduced from the regular grid can be written as :

$$\vec{\eta}_{EST} = \begin{pmatrix} \int \vec{\eta} d\sigma_{EAST} \\ 0 \end{pmatrix} = \begin{pmatrix} \eta_{x,EAST} \\ 0 \end{pmatrix}$$

$$\vec{\eta}_{NORTH} = \begin{pmatrix} 0 \\ \int \vec{\eta} d\sigma_{NORTH} \end{pmatrix} = \begin{pmatrix} 0 \\ \eta_{y,NORTH} \end{pmatrix}$$

so that EAST and NORTH elementary fluxes are simply computed in the following manner :

$$\Phi^{vis}_{EAST} = \eta_{x,EAST} R(W_{i+\frac{1}{2},j})$$

$$\Phi^{vis}_{NORTH} = \eta_{y,NORTH} S(W_{i,j+\frac{1}{2}})$$

4.3 Boundary conditions

The no-slip condition is enforced after all fluxes have been accumulated and a new solution has been computed. The velocity components are set to the prescribed values, the density is left unchanged and the total energy per unit of volume is finally modified assuming that the wall temperature is known :

$$E_{wall} = \rho C_v T_{wall}$$

4.4 Time integration

Time integration is performed using the same multi-step algorithm as described in paragraph (3.4).

4.5 General algorithm and numerical results

Unsteady solutions of Navier-Stokes equations are obtained using the following algorithm :

loop

  /* Loop on the sub-steps of Runge-Kutta */
  loop

    Get and store EAST and NORTH states
    Compute and store EAST and NORTH diffusive fluxes
    Compute and store nodal gradients
    Compute and accumulate EAST and NORTH convective fluxes
    Compute new solution

  until *nb-sub-steps=max-sub-steps*

until *t=tmax*

Memory storage defined in the case of Euler's equations is quite unchanged, we just had the geometric definition of the auxiliary cell used to compute the diffusive fluxes. On the other hand, the above algorithm involves one other step of communication when computing the auxiliary average values. But this can be done quite efficiently by combining communication and computation in the same routine call.

As a test problem, we consider the viscous flow in a rectangular domain of dimensions $[0,1]\times[0,1]$ (see Figure 11).

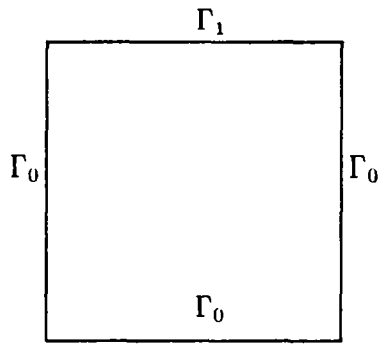

Figure 11 : Flow domain for test problem

The boundary and initial conditions are given below, the internal points being initialized with the values corresponding to Γ_0.

$$\begin{cases} u_0 = 0 \quad v_0 = 0 \quad T_0 = T_{wall} \\ \\ u_1 = 1 \quad v_1 = 0 \quad T_1 = T_{wall} \end{cases}$$

This problem has a steady solution nevertheless we want to emphasize how the computation of diffusive fluxes affects the CPU times and efficiencies obtained in the case of inviscid flows. We use the Roe's flux vector differencing for the convective fluxes which is well suited to the computation of viscous flows (see van Leer and Al. [15]). Tables 14 et 15 give results obtained on the CM-2

and the CRAY-2 mono-processor. Each computations are essentially comparable to the corresponding ones in Tables 10 and 11 of paragraph 3.5 as we have fixed the same number of time steps.

$Nx \times Ny$	VPR	$Iter$	CPU	$MFLOPS$
64 × 128	1	53	4.44	156
128 × 128	2	59	8.42	183
256 × 128	4	103	26.1	206
256 × 256	8	121	57	222
512 × 256	16	218	194	235
512 × 512	32	250	424	247
1024 × 512	64	469	1549	254

Table 14 : Navier-Stokes computations on the CM-2

$Nx \times Ny$	$Iter$	CPU	$MFLOPS$
64 × 128	53	9.65	82
128 × 128	59	21	84
256 × 128	103	74	84
256 × 256	121	170	85
512 × 256	218	618	85
512 × 512	250	1414	85

Table 15 : Navier-Stokes computations on the CRAY-2

Let's consider the values of last line of Table 14 and compare them with the corresponding ones of Table 9 obtained in the inviscid case. We note that the CPU time increase for 22.1% while the augmentation of the performance is of 6.4%. On the other part, we can predict a value of 2.0 GFLOPS on the complete configuration. Repartition of floating point operations and communications in the main steps of the algorithm above are given below.

(20)
$$\begin{array}{cc} \textit{Computation of the nodal gradients :} & 40 \\ \textit{Computation of the EAST and NORTH convective fluxes :} & 344 \\ \textit{Computation of the EAST and NORTH diffusive fluxes :} & 114 \\ \textit{Computation of the new solution :} & 34 \end{array}$$

	Get EAST and NORTH states :	8
	Computation of the nodal gradients :	8
(21)	*Computation of the EAST and NORTH convective fluxes :*	8
	Computation of the EAST and NORTH diffusive fluxes :	12
	Computation of the new solution :	8

We can see that communication steps represent about 9% of the overall work. Finally we give the values of the computational efficiency in Table 16 below.

$Nx \times Ny$	$T_{execution}$	$T_{computation}$	$Ec\%$
64 × 128	4.44	3.22	72.5
128 × 128	8.42	6.45	76.6
256 × 128	26.1	20.9	79.9
256 × 256	57	47	82.4
512 × 256	194	165	85
512 × 512	424	368	86.9
1024 × 512	1549	1367	88.2

Table 16 : Computational efficiency Ec

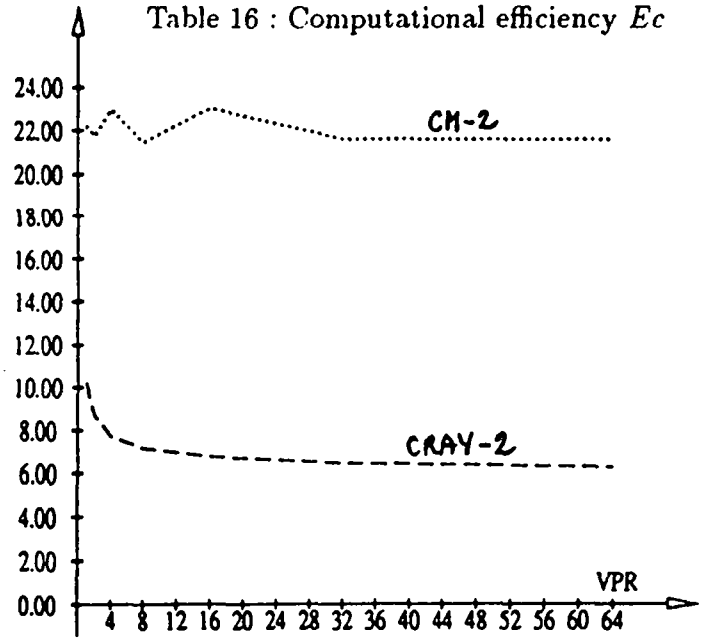

Figure 12 : CPU cost for one time step per point versus VPR

In Figure 12 above we give the evolution of the CPU cost for one time step per computational point. The dashed curve is obtained for the CM-2 with 8K

physical processors. These results show that the CM-2 is about 3.5 times faster that the CRAY-2; furthermore we can expect a ratio of 28 on the complete configuration. This last ratio may be compared with the corresponding one in the study of Long et al. [5].

5 Conclusion

In this report, we have described the implementation on the Connection Machine of a M.U.S.C.L. method for solving Euler and Navier-Stokes equations on structured grids. Several computations were made with two commonly used schemes for the discretization of convective fluxes, namely the flux vector splitting of van Leer and the flux vector differencing of Roe. Discretization of the diffusive fluxes was realized using the classical second-order accurate central differencing scheme with the constraint to be consistent with the finite volume formulation.

According to the presented set of results, it seems reasonnable to conclude that the Connection Machine is well suited to large-scale problems that characterize Computational Fluid Dynamics on structured grids. Even though we have limited our study to small size problems, we have shown that the CM can outperform a vector computer such as the CRAY-2; as the problem size increases, the CM is likely to achieve faster results. However computations on the CRAY-2 were done in double precision and in mono-processor mode.

On the development point of view, we have emphasized that the use of the PARIS library is necessary to achieve high performance rates. Nevertheless this leads to a low level style of programming that is quite tedious.

We have limited ourselves to computations on a simple geometry easily discretized with the use of stuctured grids. Inter-processor communications were efficiently performed with the NEWS grid facilities and we have verified that communication costs do not degrade the computational efficiency when larger set of data are manipulated. At this point, we have to remark that our computations doesn't make use of STENCIL mechanisms. A STENCIL routine is a very efficient software primitive that combines communication steps with different neighbors (in the NEWS topology) and linear combination of the corresponding values in the same routine call. For instance, we could have used such mechanisms in the computation of the nodal gradients (see expression (10)). At high VPR the communication time in our computation doesn't exceed 10% of the total execution time, so we don't think that the use of STENCILS would have resulted in a noticeable increase of the computational efficiency. However, real life problems involve complex geometries hardly handled in the same way. On the other hand finite element methods on unstructured grids are well suited to the simulation of such flows. Future extensions of our work will include the resolution of full Navier-Stokes equations using a mixed finite element/finite volume method on unstructured grids. For such methods inter-processor communications will represent the main obstacle to high perfomance rates while the computational part remains quite unchanged from what has been depicted in

this report.

However, this study was primarly motivated by direct simulation methods applied to the modelisation of strong unsteady phenomena that can be modelled using the Navier-Stokes equations. Here we have just emphasized the suitability of the Connection Machine system to such computations; so in a future report we shall present some applications of this code to more realistic problems.

Références

[1] *Connection Machine Model CM-2 Technical Summary*, Thinking Machines Corporation, Version 5.1 May 1989

[2] HILLIS D., *La Machine à connexions*, Masson Editions, (1988)

[3] McBRYAN O. A., *New Architectures : Performance Highlights and New Algorithms*, Technical Report CU-CS-403-88, (1988)

[4] JESPERSEN D. C. - LEVIT C., *A Computational Fluid Dynamics Algorithm on a Massively Parallel Computer*, AIAA paper 89-1936, (1989).

[5] LONG L. N. - KHAN M. M. S. - SHARP H. T., *A Massively Parallel Three-Dimensional Euler/Navier-Stokes Method*, AIAA paper 89-1937, (1989).

[6] AGARWAL R. K., *Development of a Navier-Stokes Code on a Connection Machine*, AIAA paper 89-1938, (1989).

[7] SAATI A. - BIRINGEN S. - FARHAT C., *Solving Navier-Stokes Equations on a Massively Parallel Processor : Beyond the 1 GFLOPS Performance*, Intern. Journ. of Supercomputer Applications, Vol 4, No 1, (1990).

[8] ANDERSON D. A. - TANNEHILL J. C. - PLETCHER R. H. , *Computational fluid mechanics and heat transfer*, Hemisphere Publishing Corporation, Mc Graw-Hill Book Company, (1984).

[9] LOMAX H. - KUTLER P. - FULLER F.B., *The Numerical Solution of Partial Differential Equations Governing Convection*, AGARD-AG-146-70, (1970)

[10] DESIDERI J. A. - GOUDJO A. - SELMIN V., *Third-Order Numerical Schemes for Hyperbolic Problems*, Rapport de Recherche INRIA No 607, (1987).

[11] FEZOUI L. - STEVE H., *Décomposition de Flux de van Leer en éléments finis*, Rapport de Recherche INRIA No 830, (1988).

[12] ROE P. L., *Approximate Riemann Solvers, Parameters Vectors and Difference Schemes*, J. Comp. Phys., **43**, pp. 357-371, (1981).

[13] VAN LEER B., *Flux-Vector Splitting for the Euler Equations*, Lecture Notes in Physics, Vol 170 (1982).

[14] VAN LEER B., *Towards the Ultimate Conservative Difference Scheme V : a Second-Order Sequel to Gudonov's Method*, Journal of Computational Physics, Vol 32, (1979).

[15] VAN LEER B. - THOMAS J. L. - ROE P. L. - NEWSOME R. W., *A Comparison of Numerical Flux formulas for the Euler and Navier-Stokes Equations*, AIAA paper 87-1104, (1987).

[16] McCORMACK R. W. - BALDWIN B. S., *A Numerical Method for Solving the Navier-Stokes Equation with Application to Shock-Boundary Layer Interactions*, AIAA paper 75-1, (1975).

[17] CHAKRAVARTHY S. R. - SZEMA K. Y. - GOLDBERG U. C. - GORSKI J. J., *Application of a New Class of High Accuracy TVD Schemes to the Navier-Stokes Equations*, AIAA paper 85-0165, (1985)

[18] LIOU M. S. - HSU A. T., *A time Accurate Finite Volume High Resolution Scheme for Three Dimensional Navier-Stokes Equations*, AIAA paper 89-1994, (1989)

[19] HARTEN A. - LAX P. D. - VAN LEER B. , *On Upstream Differencing and Godunov-Type Schemes for Hyperbolic Conservation Laws*, SIAM Review, Vol 25, No. 1 (1983)

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique.

ISSN 0249-6399